FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

DIGITAL LOGIC SIMULATOR COMPARISON IN EDUCATIONAL PURPOSE

Ville Savolainen

Supervisor: Jukka Lahti

DEGREE PROGRAM IN ELECTRICAL AND COMMUNICATIONS ENGINEERING
2020

Savolainen V.S. (2020) Digital logic simulator comparison in educational purpose. University of Oulu, Degree Program in Electronics and Communications Engineering. Bachelor's Degree, 23 pages

ABSTRACT

This work is comparing browser-based and computer installed digital logic circuit simulators, which can be used in for learning digital logic. Theory section introduces basics of combinational and sequential logic.

Key words: Digital logic simulator, Logisim, CircuitVerse.

Savolainen V.S. (2020) Digitaalipiirisimulaattorien vertailu opetuskäytössä. Oulun yliopisto, Elektroniikan ja tietoliikennetekniikan tutkinto-ohjelma. Kandidaatintyö, 23 sivua

TIIVISTELMÄ

Kandidaatin tutkielmassa verrataan Internet-selaimissa toimivia pohjaisia ja tietokoneelle valmiiksi asennettuja digitaalipiirien logiikkasimulaattoreita opetuskäytössä, huomioiden suorituskyky, helppokäyttöisyys ja eri ominaisuuksia logiikkapiirien kuvaamiseen. Teoriaosiossa esitellään kombinaatio-, sekä sekvenssilogiikan perusteita.

Avainsanat: Digitaalitekniikka, Logisim, CircuitVerse.

TABLE OF CONTENT

FOREWORD

Digital logic has revolutionized modern world and will thrive possibly for decades. To learn about these complex digital systems, one needs to understand the basics of digital theory. This work is made to ease this learning.

Oulu, 06.03.2020

Ville Savolainen

1. INTRODUCTION

When one is learning about digital logic circuits, there are many simulators which allow building digital circuits and verifying their functionality. Enterprises use complex simulators for designing and testing FPGA- and ASIC-circuits. These simulators are efficient since they have many different features, but they are complex to use. Therefore, they are not best for educational purposes on early stages of learning the basics of digital logic circuits. Nowadays there are many digital logic circuit simulator software for learning purposes. These simulators must be visually clear and easy to use but still efficient enough to simulate basic circuits. Their efficiency is not on a par with enterprise software and their functionality is often connected to user's computer hardware efficiency.

This research compares an internet browser-based simulator and a computer-based simulator software for learning purposes. Computer-based simulator often needs performance from computers hardware. Browser-based simulators use servers for circuit calculations. Thus, simulator efficiency does not need so much from user's computer.

Chosen features for comparison were ease of use, efficiency and testing circuits. Ease of use was subcategorized by getting started, menu structure and building circuits. Efficiency comparison was done by comparing different features and limitations of software. Essential features for circuit testing in this research were testbench, logging and verification.

2. THEORY

According to Harris D. and Harris S., a digital circuit is a module with discrete-valued inputs and outputs and a specification describing the function and timing of module. [1] Digital circuits are used for processing digitized signal which is a quantized analog signal.

Analog signal in electronics is a continuous signal that represents data by voltage and current signal. Quantizing means sampling of analog signal so that it is represented in discrete-valued variables described by Harris D. and Harris S. in their book [1]. 0 means that signals voltage level is under defined threshold level and 1 that voltage is over threshold level. In most situations, if a signal's voltage level is under threshold level, signal is attenuated. Then it can be considered that there is no voltage which means there is no signal. Over threshold signal is amplified, so that the level between on- and off -states is clearer. These two states can also be represented in base-2 numeral system or in binary system.

Most common tools for describing signal states in digital electronics are Boolean algebra, truth tables, timing diagrams and circuit diagrams. Boolean algebra is a mathematical language, used for describing arithmetic of two-state systems [2]. More information about Boolean algebra can be found in section 2.1.

Crow J. describes truth tables as mathematical tables that are used to show how the output(s) depends upon the input(s) [2]. These two states in truth tables indicate if the signal is TRUE or FALSE (1 or 0). This is where the name "truth table" comes from. Variables in this work are represented by symbols A, B, C, Y and X. A-, B- and C-symbols as input signal, Y- and X-symbol as the output signal. Timing diagrams are used for representation of the signal in the time domain and a circuit diagram is a visual display of an electrical circuit.

2.1. Boolean Algebra

In Boolean algebra, a signal is modified by three operators [2]. These operators are called AND-, OR- and NOT-operators. NOT-operator takes only one variable as input, AND- and OR-operators take number of variables as input, but all operators produce one output. Operator symbols can be found found in Figure 2.1.

Figure 2.1. From left to right: AND-, OR- and NOT-operators in CircuitVerse-simulator.

NOT-operator (Table 2.1.) inverts the input so that output is a complement of input. [2]

Table 2.1. NOT-operator truth table

A	Y
1	0
0	1

OR-operator sets output to 1 if one of the inputs is 1. AND-operator sets output to 1 if all of the inputs are 1. Truth tables of the operators can be found in Table 2.2. [2]

Table 2.2. OR- and AND-operator truth table with 2 inputs

A	B	OR-output	AND-output
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

Modified operators are Exclusive OR (later XOR), Not OR (NOR), Not AND (NAND) and Not XOR (XNOR).[2] These operators are compiled by adding NOT-operator to previously mentioned operators, except XOR (Figure 2.3.) which consist of NAND-, AND- and OR -operators. Operator symbols are presented in Figure 2.2. and their truth tables can be found in Table 2.3.

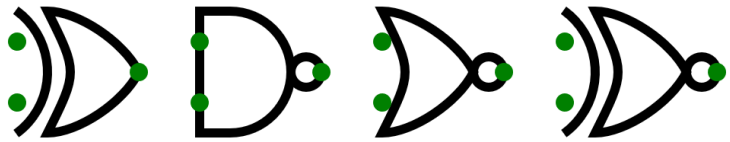


Figure 2.2. From left to right: XOR-, NAND-, NOR and XNOR -operators in CircuitVerse-simulator.

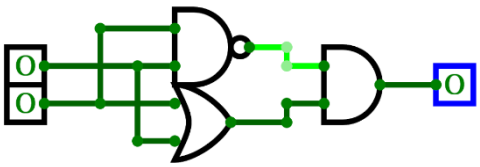


Figure 2.3. XOR-operator described in gate-level by CircuitVerse -simulator.

Table 2.3. XOR-, NAND-, NOR- and XNOR -operator truth table

A	B	XOR-output	NAND-output	NOR-output	XNOR-output
0	0	0	1	1	1
0	1	1	1	0	0
1	0	1	1	0	0
1	1	0	0	0	1

In Boolean algebra, there are theorems for single variable equations. They are idempotent law, involution law, inverse elements and properties of identity elements. Truth tables of these theorems are described in Table 2.4. Idempotent law states that if a variable is operating itself, it goes to all inputs. Inverse elements describe that when a variable is operated with its complement, output is 0. According to involution law when same variable is operated twice with same operator, variable returns to its original state. Properties of the identity elements express the information by giving output in numeral form.[1]

Table 2.4. Boolean algebra single variable theorem in visual format

Idempotent law	Inverse elements	Involution law	Properties of identity elements
A * A = A	A + $\bar{A}$ = 0	A = $\overline{\overline{A}}$	A * 0 = 0
			A * 1 = 1
A + A = A	A * $\bar{A}$ = 0		A + 0 = 1
			A + 1 = 1

In most cases there are multiple variables and therefore more rules and laws, such as commutative law, associative law, distributive law and De Morgan's theorem, which affect the equations. Equations of these theorems can be seen in Table 2.5. Commutative law states that when operating AND- or OR-operations together, order is insignificant. Associative law shows how variables are grouped together. Distributive law tells how to expand equations out. De Morgan's theorem defines that complementing the result of the all OR variables together is equivalent to the complements of individual AND variables and vice versa. [1]

Table 2.5. Boolean algebra multiple variable theorem in table format

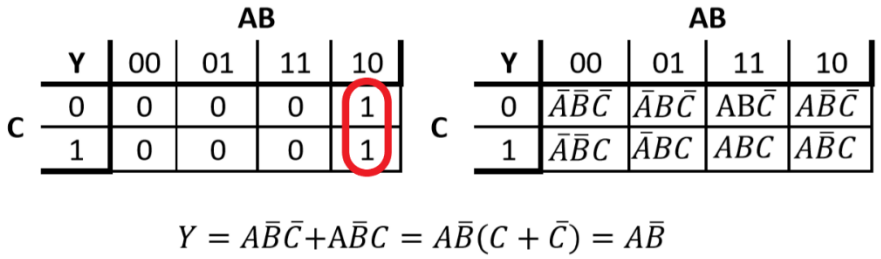
Commutative law	Associative law	Distributive law	De Morgan's theorem
A*B=B*A	(A*B)*C=A*B*C=A*(B*C)	A*(B+C)=A*B+A*C	$\overline{A+B} = \overline{A} * \overline{B}$
A+B=B+A	(A+B)+C=A+B+C=A+(B+C)	A+(B*C)=(A+B)*(A+C)	$\overline{A*B} = \overline{A} + \overline{B}$

According to Harris D and Harris S, Karnaugh maps are "graphical method for simplifying circuit" [1]. Simplifying of the logic with this method makes it possible to visualize the logic. The method puts the variables next to each other in "map" so that the variables can be combined.

Example 1: Output is 1 if A is 1, B is 0 and C is 0 or A is 1, B is 0 and C is 1. Truth table of this is visualized in Table 2.6. Karnaugh map in this example shows an easier solution. Generally, in Boolean algebra the variables would have to be opened and combined but, in this example, they are marked with a red circle in Figure 2.4. Karnaugh maps do not always ease optimizing, but they can be a great help compared to truth tables.

Table 2.6. Example 1 situation truth table of three inputs (A, B and C) and one output (Y)

A	0	0	0	0	1	1	1	1
B	0	0	1	1	0	0	1	1
C	0	1	0	1	0	1	0	1
Y	0	0	0	0	1	1	0	0



$$Y = A\bar{B}\bar{C} + A\bar{B}C = A\bar{B}(C + \bar{C}) = A\bar{B}$$

Figure 2.4. Top left is a Karnaugh map from Example 1, top right the same Karnaugh map with different presentation and below is the opened and simplified equation.

Example 2: Using Boolean algebra to simplify Boolean expression: [1]

Y=ABC+ABC+ABC+ABC=AB(C+C) +AB(C+C) =AB+AB=A(B+B) =A

It is important to minimize the logical expression to its simplest form, so that the circuit afterwards is smaller. In minimization Examples 4 AND-gates, 2 input signal and 1 OR-gate were removed (Figure 2.5.).

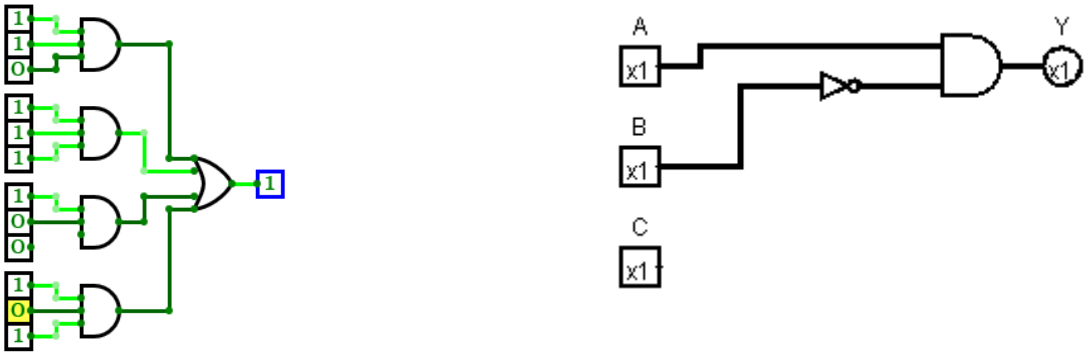


Figure 2.5. An equation with no minimisation in CircuitVerse (left) and minimized equation in Logisim (right).

2.2. Combinational and sequential logic

Generally digital circuits are classified in two categories: combinational or sequential.
The main difference between these two circuits is memory. Combinational circuit outputs depend only on current input values at that time [1], which means they are memoryless. Outputs of sequential circuits depend on input sequence [1], meaning circuit functionality takes both current and previous input values and therefore it has a memory. Examples of sequential logic circuits are flip-flops and RAM-memory.

2.2.1. Combinational logic

Combinational logic is used in digital circuits to transfer, process or transform data. Previously mentioned logic gates are combinational. Other example of combinational logic are multiplexers, decoders, adders, parity checkers and comparators. The following are modelled these examples.

Multiplexers in electronics are devices which provide a way of selecting one out of many digital input signals through Select signal (Figure 2.6.) [2]. Demultiplexer works the other way around converting a single input to one of the outputs selected by control line.

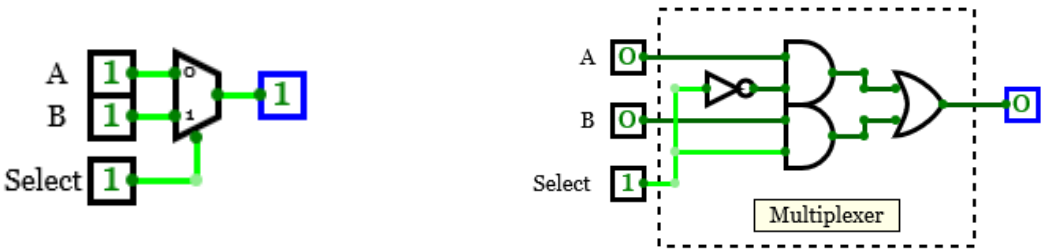


Figure 2.6. Simple 2 input multiplexer visualized by operators (left) and as block (right) in CircuitVerse -simulator.

Decoder sets one of the output signals high, depending on input signal (Figure 2.7.). Encoders do reverse operation compared to decoder setting one of the input signals to output. This conversion is called coded binary, s-bit [2]. Truth table of 2-to4 decoder is visualized in Table 2.7.

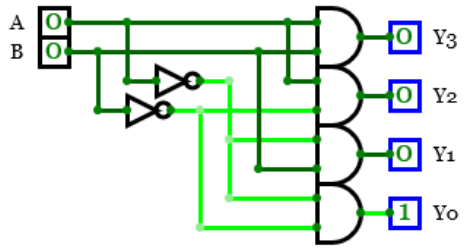


Figure 2.7. 2-to-4 decoder visualized by operators in CircuitVerse-simulator.

Table 2.7. Truth table of Figure 2.8. 2-to-4 decoder

A	B	Output
0	0	Y0
0	1	Y1
1	0	Y2
1	1	Y3

Comparator compares input signals through XOR-logical gate: only one of the inputs is shown in output [2]. Parity generator and checker work as error correctors. Generator adds extra bit(s) to data and checker checks if there are any additional bits in line. If there are, a corruption has happened in data processing.

Adder is a calculator that performs addition and subtraction to input data. There are two kind of adders: full adder and half adder. Full adder differs from half with carry-bit. In full adder, carry bit is included to input from previous calculation stage [1]. Half adder is carryless which means there is carry-bit in input. Circuit of full adder and half adder are visualized in Figure 2.8. and their truth tables are presented in Table 2.8.

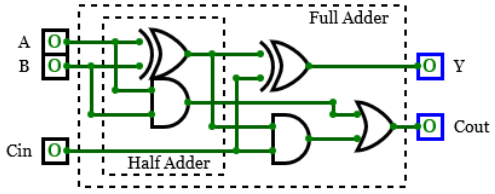


Figure 2.8. A Full and a half adder visualized by CircuitVerse.

Table 2.8. Truth table of a full adder from Figure 2.8. A, B and $C_{in}$are inputs. C_{in} is carry-bit in, and C_{out} is carry-bit out. Y is an output

A	B	C_{in}	C_{out}	Y
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	0

2.2.2. Sequential logic

According to Crowe J. sequential circuits are essentially combinational circuits with feedback [2]. This means that circuit has memory which remembers its present state and that state is placed in combinational logic input (Figure 2.9.).

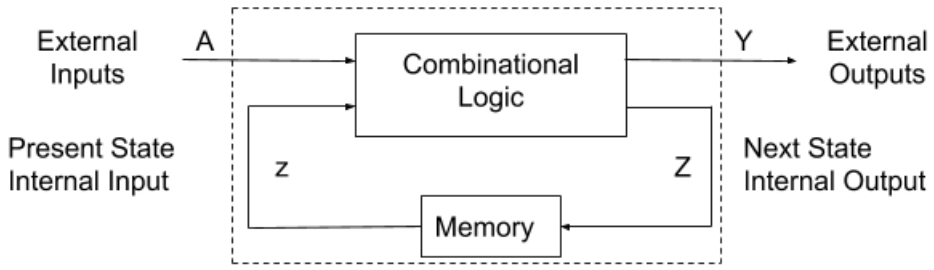


Figure 2.9. Visual representation of sequential logic.

Latches and flip-flops are simple examples of sequential logic. They both store one-bit state variable from previous cycle, but when a circuit is turned on the state of

the state variables cannot be predicted. Inputs are cross coupled gaining fundamental element of memory block: they are bistable, an element with two stables states [1].

The two most simple latches are SR-latch and D-latch. SR-latch works with two cross-coupled inputs (Set and Reset), which are connected to 2 NOR-gates (Figure 2.10.). The problem with SR-latch is that it behaves unpredictably when both inputs are asserted simultaneously [1]. This causes that the change in input is reflected in the output immediately. To solve this problem D-latch is developed. (Figure 2.10) D-latch tells the output when to change. Truth table of D latch is represented in Table 2.9.

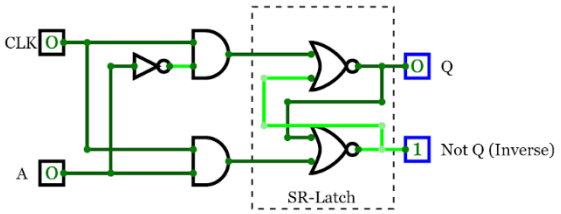


Figure 2.10. D-latch visualized with SR-latch by CircuitVerse. A is an input, CLK is a clock of circuit and Q is an output.

Table 2.9. Truth table of D-latch from Figure 2.10. A is an input, CLK is the clock of the circuit, $\bar{A}$ is inverse from A, S and R are intermediate results, Q is an output and $\bar{Q}$ is inverse value from Q. Prev-subscript describes the value of the previous state.

A	CLK	$\bar{A}$	S	R	Q	$\bar{Q}$
X	0	X	0	0	Q_{prev}	$\bar{Q}_{prev}$
0	1	1	0	1	0	1
1	1	0	1	0	1	0

A latch is a level-triggered circuit whereas flip-flops are edge-triggered circuits. The output of the latch activates when the level of input or clock changes. A flip-flop output signals changes with inputs positive or negative edge. That is why it is also known as an edge-triggered flip-flop [1]. Difference in timing diagram of D-latch and D-flip-flop is presented in Figure 2.11.

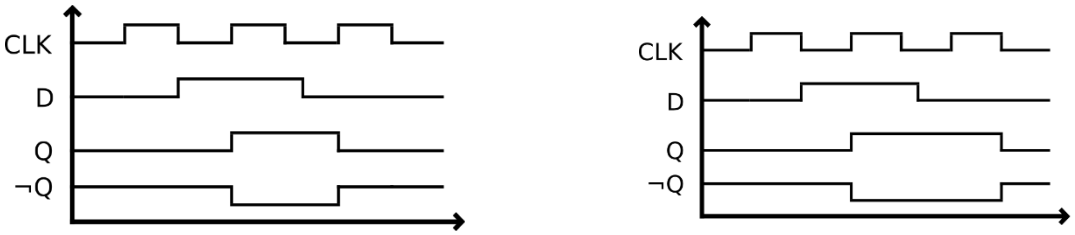


Figure 2.11. Timing diagram of D-latch (left) and D-flip-flop (right). CLK is clock of the circuit, D is an input, Q is an output and ¬Q is inverse of Q.

3. SIMULATOR COMPARISON

Nowadays logic circuits can be simulated by countless logic simulators. For educational purposes, simulator-software should be visualizing, free and simple to use. In this work a browser -based online simulator and a computer -installed simulator are compared. CircuitVerse online simulator and Logisim 2.7.1 computer installed simulator were chosen for comparing.

3.1. Ease of use

Getting started with software is easier with browser based CircuitVerse. User needs to only open the webpage and start building a circuit. There is no need for creating account and logging in even though it is possible. Logging in enables saving projects and circuits and publishing them to other users. [3] Logisim -program has to be downloaded but there is no need to install it on computer because downloaded file is in .exe-format. In other words, it does not require additional disk space, but it has to be installed every time it is opened. Logisim software however needs read and write permissions or administrator login. Offline use is possible in Logisim but not in CircuitVerse.

At first glance colour world seems to be more pleasant in CircuitVerse, as the main colour of Logisim is gray. Both of the programs share same elements for creating circuit which can be seen in starting menus presented in Figure 3.1. Almost all circuit building blocks can be found in different menus.

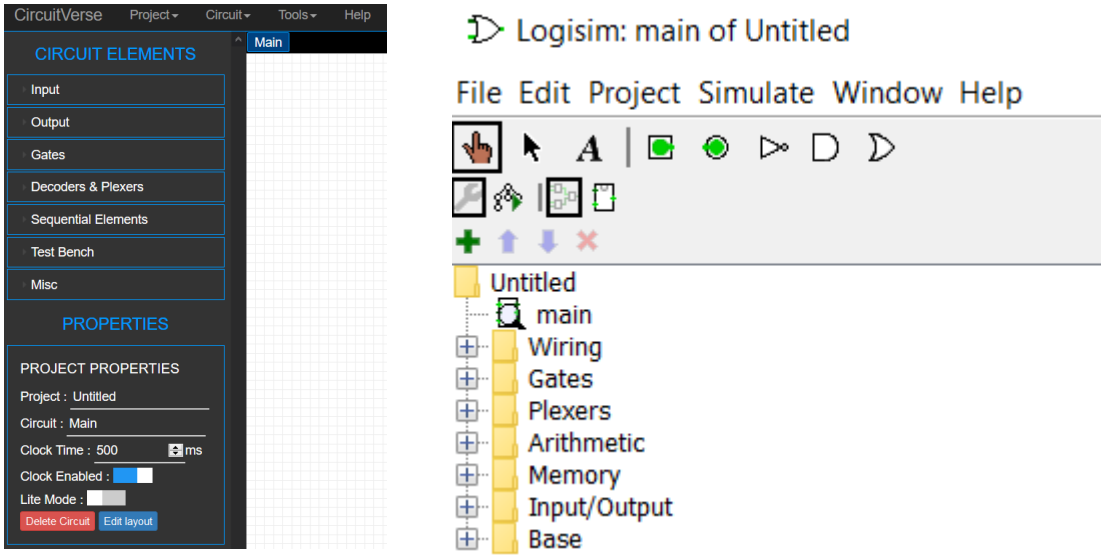


Figure 3.1. CircuitVerse-simulator starting menu (left) and Logisim simulators starting menu (right).

Example circuits for CircuitVerse are more accessible and found in front page. There are other users published circuits, editor pics of circuits and featured examples.

Example circuits for Logisim are found in internet and it can open circ-format libraries and -circuits, which isn't possible in CircuitVerse. However, it is possible to open and simulate other users' published circuits in CircuitVerse.

There are two ways of starting to build circuit: manually or by Boolean logic table. When building a circuit by hand, designer starts by adding input and output pins, combinational elements, such as decoders, multiplexers and logical gates. Sequential elements are also possible to add. For example, different flip flops, latches and memory block. After positioning blocs in schematic, user connects them with wires. Logisim has also build-in mathematical arithmetic's, counters and registers. Logisim has a feature for combinational analysis: if circuit is done by hand, it will draw a Boolean logic table, a simplified Karnaugh map and a mathematical expression of circuit (Figure 3.4.). [4]

Simulator can generate circuit from Boolean logic table. Using this feature is started by adding input(s), output(s) and describing circuits functionality by adding output values in different input values. An example circuit is presented in Figure 3.2. and the circuit is generated in Figure 3.3. Logisim has an option for using two-input gates and/or NAND-gates only for optimization. Boolean table generated circuits are only available for combinational circuits.

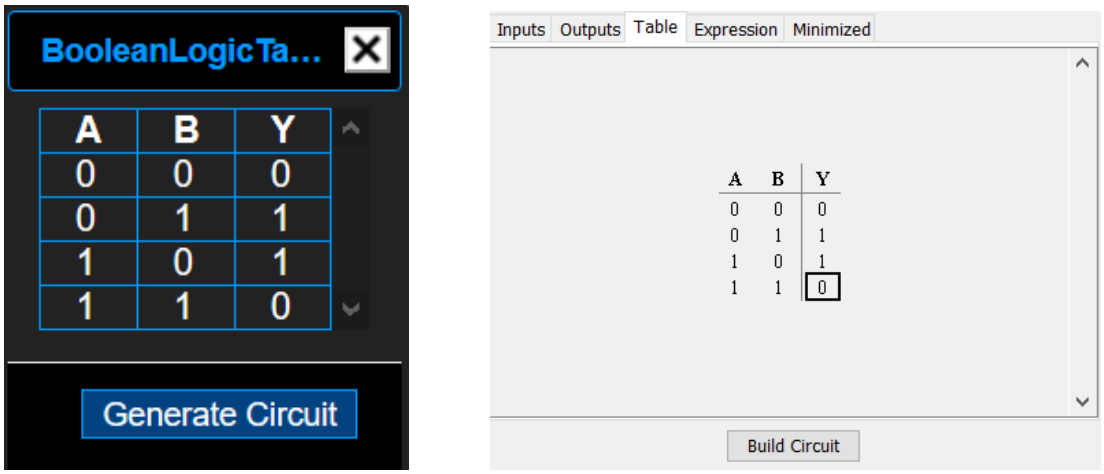


Figure 3.2. On the left CircuitVerse and on the right Logisim Boolean logic table circuit generator.

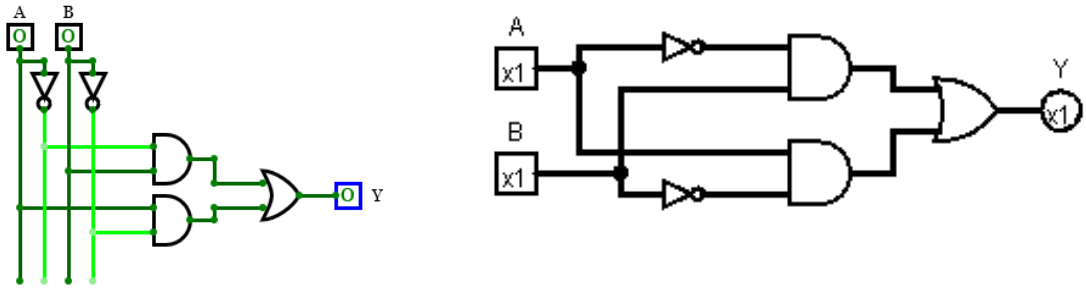


Figure 3.3. CircuitVerse (left) and Logisim (right) Boolean generated example circuit from Figure 3.2.

Figure 3.4. Logisim expression (left) and Karnaugh map (right) of Figure 3.2. circuit.

3.2. Efficiency

Comparable elements of efficiency are limitations and features. Comparison of the key elements between simulators is listed in Table 3.1.

Table 3.1. Limitation and feature comparison between Logisim and CircuitVerse

Compared element	CircuitVerse	Logisim
Max clock frequency	4.1 kHz	20 Hz
Input/output max bit width	32-bit	32-bit
Logic gate max bit width	32-bit	32-bit
Logic gate max inputs	32-inputs	10-inputs
Multiplexer max bit width	32-bit	32-bit
Multiplexer select bits	5-bits	~12-bits, goes notably slower over 12-bits
Three-state behaviour enable/disable	Yes	No, outputs have unknown state, but it can't be modified

Combinational simulation is a basic function for logic circuits, but sequential simulation is also possible for both simulators. Simulators share built-in sequential elements by flip flops, RAMs and ROMs, providing D-, T-, S-R- and J-K flip flops. They differ in D-latches which CircuitVerse only has. Register, shift register and counter blocks which are only in Logisim but not in CircuitVerse. Building a circuit by Boolean table was introduced in section 3.1, but it only allows building combinational logic circuits. Sequential logic Boolean table circuit generation is not possible because the signals can have previous states [4].

3.3. Testing and verification

Both simulators provide test input signals button(s), zero or x (unknown state), teletypewriter (TTY), keyboard, clock, ground (constant 0), power (constant 1) and input pins which can be made to point 1. Logisim has 8-bit width joystick and transistors but CircuitVerse does not have these. CircuitVerse however can import input signals from testbench.

CircuitVerse allows creating a testbench for circuits (Figure 3.5.). [3] This feature works by adding inputs and wanted outputs and therefore it is possible to create different cases. After importing these input and output testbenches to simulator it tells if the simulation is working properly or not.

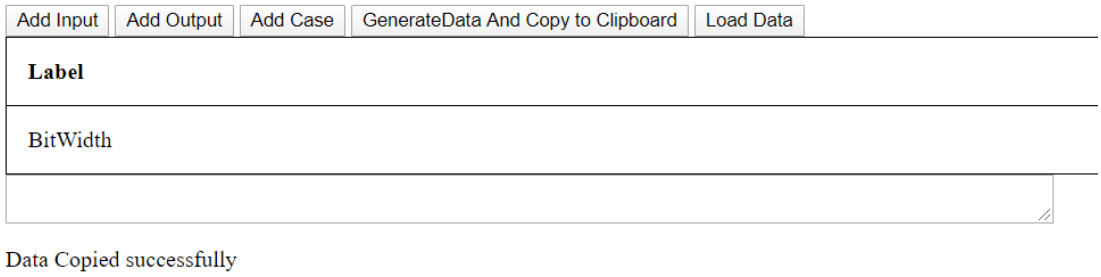


Figure 3.5. CircuitVerse testbench appearance.

Logisim can simulate circuits in command-line by for example java. This feature intends to help in scripted verification and automated testing. [4] Example found in Logisim documentation is visualized in Figure 3.6. and command-line terminal output in Table 3.2.

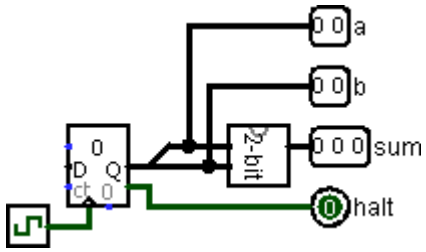


Figure 3.6. Example circuit by Logisim.

An example of command-line command for circuit presented in Figure 3.6. is:

java -jar logisim-filename.jar adder-test.circ -tty table

, where java opens java simulator, -jar command tells which .jar-library will be used, circ-file will be Logisim simulation design, -tty command tells wanted output, which is in this case a table. Output in documentation example is shown in Table 3.2.

Table 3.2. Output Example shown in command-line by Logisim command-line use. The first two columns from left are 2-bit input signals in each situation and the third row is a 3-bit output of an adder

00	00	000
01	00	001
10	00	010
11	00	011
00	01	001
01	01	010
10	01	011
11	01	100
00	10	010
01	10	011
10	10	100
11	10	101
00	11	011
01	11	100
10	11	101

Each logical gate increases delay of the signal. Propagation delay is the time for signal to change and become stable from input to logical gates output. Critical path is the longest path of the signal from input to output. Previously mentioned minimization of logic is also critical in timing specification of combinational circuit because it decreases number of logical gates in circuit and therefore decreases the delay of signal. This is visualized in Figure 3.7.

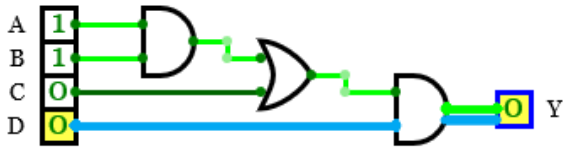


Figure 3.7. Critical path is bright green and the shortest path is turquoise in CircuitVerse.

Timing constraints in CircuitVerse can be modified by adding delay on each component. “Logisim will add a delay to a component's propagation. This is intended to simulate the unevenness of real circuits.” [4] This random delay can be disabled from options in Logisim but cannot be modified. Logisim also calculates critical and shortest path of each circuit (Figure 3.8.).

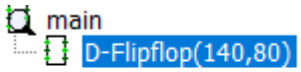


Figure 3.8. Critical path (left) and shortest path delay (right) in Logisim.

CircuitVerse has Flag-objects, where different flags can be put in signal path. Then the program shows values of the signals with time as x-axis (Figure 3.9.). For advanced constraints, such as pulse width, setup or hold cannot be modified.

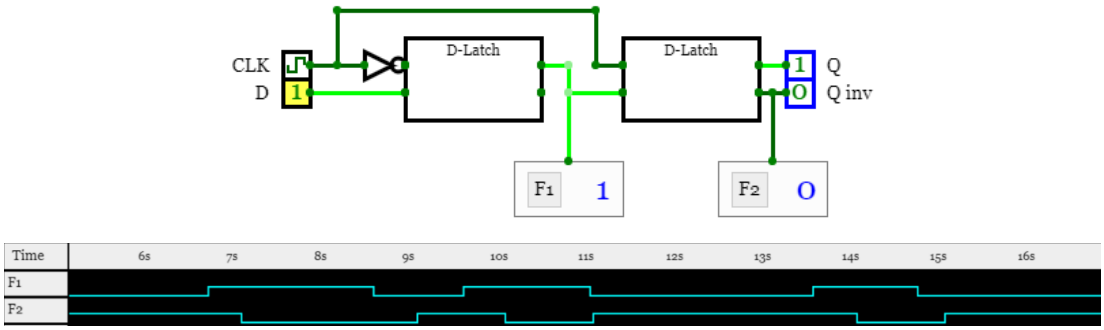


Figure 3.9. D-flip flop circuit and lower flag output in timeline by CircuitVerse.

Logging of the circuit functions is a basic feature for verifying that the circuit works. Logisim creates output log-files, so it can be seen later how circuit works. This enables running the circuits in command prompt. CircuitVerse does not make any logging files, so simulations must be verified in real time, by flags or creating different test cases in testbench. Logisim has therefore an advantage in automated simulations. Logisim example of logging file of 4-bit counter is presented in Figure 3.10.

A	B	C in	C out	Y
0000	0000	0	0	0000
0001	0000	0	0	0000
0001	0000	0	0	0001
0011	0000	0	0	0011
0111	0000	0	0	0111
1111	0000	0	0	1111
1111	0001	0	1	0000
1111	0011	0	1	0010
1111	0111	0	1	0110
1111	1111	0	1	1110
1111	1111	1	1	1111

Figure 3.10. 4-bit counters log-file produced by Logisim and visualized in Excel.

4. DISCUSSION

Both simulators became familiar during the study and they both performed well in all tests. Neither of the software crashed or showed unexpected behaviour. Outcome of the comparing experiment was clear and unambiguous. Logisim is better than CircuitVerse considering efficiency and circuit building elements. However, these elements do not give significant advantage. CircuitVerse performance is adequate for educational purposes. Example codes are more accessible in CircuitVerse which makes getting started easier. Conclusion of this study is that CircuitVerse is better for learning and Logisim would be chosen for advanced design of circuits.

Further study can be conducted by comparing more simulators, for example Infineon as Internet-based simulator. Other objects for the future could be an inquiry from digital technique students and adding more elements to comparison in order to achieve more comprehensive results.

5. SUMMARY

The purpose of this study was to compare Internet-browser-based and computer-based digital logic simulators. The main aspect was to use simulator for learning about digital circuits. Comparable features were ease of use, efficiency and testing circuits.

Getting started prefers CircuitVerse because the user can go to a webpage and start immediately building circuits. Logisim must be downloaded before using. Colour world is more pleasant in CircuitVerse as the main colour of Logisim is grey. Both simulators share the same elements for creating circuits. Almost all circuit building blocks can be found in both programs. Example codes for Logisim can be retrieved from internet because it can open circ-format libraries and circuits. CircuitVerse example codes are more accessible and are placed at the front page.

As the result of the efficiency comparison, both simulators seem to have similar limitations. Logisim has better efficiency when comparing other features such as registers and counter blocks. Both simulators provide test input signals from same sources. Logisim has more sources, such as 8-bit width joystick and transistors, which CircuitVerse does not have. CircuitVerse can however import input signals from testbench.

CircuitVerse allows creating a testbench for circuits. Logisim can simulate circuits in command-line. Timing constraints in CircuitVerse can be modified by adding delay on each component. Logisim adds delay to a component's propagation, but this delay cannot be modified. CircuitVerse has flag-objects which can be put in signals path. Then the program shows values of the signal with time as x-axis.

6. REFERENCES

[1] Harris, David Money, Harris, Sarah L. (2007) Digital Design and Computer Architecture. Morgan Kaufmann.

[2] Crowe, J. (1998) Introduction to digital electronics. Oxford; Boston: Newnes.

[3] CircuitVerse documentation (read 6.3.2020). URL: https://docs.circuitverse.org/#/

[4] Logisim documentation (read 6.3.2020). URL: http://www.cburch.com/logisim/docs/2.6.0/en/guide/index.html